Logical Proof Tool

כלי הוכחות לוגיות

מגישים : רועי נוביץ' 203897186

אורן אור 203985262

מנחה : פרופסור דני קוטלר

Project Github link: https://github.com/roynovich1451/Final_project_logic_calculator

תוכן עניינים

למה בחרנו בפרויקט זה

כאשר ביצענו פגישת סיעור מוחות ראשונית לגבי הפרויקט שאותו אנו רוצים לעשות היה לנו ברור שאנחנו רוצים לעשות פרויקט משמעותי, מה שייתן לנו ניסיון בתחום הפיתוח משהו שיתחלל את כל שלבי הפיתוח בתוכנה ובנוסף יוכל להיכנס ל'תיק העבודות' שלנו. כשעברנו על רשימת האופציות לנושאים ראינו את הנושא של המחשבון לוגיקה שמנחה דני. באותו שלב בדיוק סיימנו את הקורס ב .net. WPF #C עם דני ורצינו להמשיך ולעבוד עם שפת התכנות הזאת. כשדיברנו עם דני הוא הסביר שהרעיון של מחשבון לוגיקה כבר נעשה מספר פעמים במכללה. עם זאת, דני רצה שיהיה כלי שיעזור לכתוב הוכחות לוגיות, מבלי להגיש את הפתרון לסטודנט על מגש של כסף. אהבנו את הרעיון מפני שהוא דרש מאיתנו למצוא פתרון לבעיה אמיתית שחווינו בעצמנו.

ישנם כלים רבים אשר נותנים פתרון מידי להוכחה או יוצרים טבלת אמת, חלקם גם נעשו על ידי סטודנטים בפרויקטים קודמים בהנחיית דני, אך במקרה זה היה ברור כי זהו פרויקט שלא נעשה בעבר, מבחינת היקף ומבחינת קושי המימוש.

מלאכת שיעורי הבית בקורס לוגיקה מצריכה כתיבה רבה. מעצם צורת כתיבת ההוכחה על כותב התשובה להיות ברור ומדויק ולשמור על סדר. בניגוד למקצועות רבים אחרים אשר כוללים בעיקר מלל ואולי סימונים מתמטיים סטנדרטים, הוכחה לוגית מצריכה סימונים שאינם טריוויאליים וסטודנט אשר רוצה לענות על שאלות אלו בוורד (או בכל תמלילן אחר) מוצא עצמו מבזבז זמן רב על חיפוש הסימנים וארגונם בטבלה, ולכן רבים נמנעים ובוחרים להגיש בכתב יד. הגשה בכתב בסוג התרגילים זה הקשה על הסטודנט מעצם צורת ההוכחה. לדוגמה, אם לאחר סיום הוכחה סטודנט מגלה שחסרה לו שורה באמצע הוכחה הוא נאלץ לשנות את כל המבנה של טבלת ההוכחה. בנוסף לכך ישנם סטודנטים בעלי כתב לא מובן או קושי בכתיבה מסודרת וברורה, מה שמקשה באופן ישיר גם על בודק התרגילים של הקורס.

רצינו לייצר תוכנה אשר תוכל לפתור את בעיות אלו ולתת ממשק עבודה נוח ופשוט לסטודנטים של הקורס

מטרות הפרויקט

- **מטרה עיקרית:** יצירת כלי נוח לכתיבת ובדיקת הוכחות בלוגיקה, בין השאר בשיעורי הבית והתרגילים בקורס.
- **מטרות משנה:**
  - יצירת ממשק עבודה על הוכחות לוגיות שיהיה כמה שיותר נוח.
  - מימוש בדיקת חוקי הלוגיקה בתוכנה.
  - יצירת תוכנה שמשרתת את הסטודנט אך בו בעת לא מאפשרת לו גישה ישירה לפתרון התרגיל.
  - בניית הפרויקט כ- Open source לצורך תיקונים עתידיים והרחבה.
  - ממשק טעינה/שמירה אל מול Word.

רקע מתמטי

במתמטיקה הוכחה היא סדרה סופית של טענות הנובעות זו מזו בעזרת כללי היסק, תוך שימוש בהגדרות, אקסיומות, ובידע קודם שהוכח קודם לכן, המראה שטענה מסוימת היא נכונה. בלוגיקה מתמטית ,הוכחה היא סדרה סופית של פסוקים במסגרת שפת תחשיב יחסים נתונה, המורכבת מאקסיומות ומגזירות באמצעות כלל היסק.

מבנה של הוכחה לוגית הוא טענה ונימוק, כאשר בקורס לומדים שניתן למספר את השורות ולרשום בעמודת הנימוק את השורות שבהן הוא נמצא.

כלל הגזירה "$\wedge$ elimination":

כלל הסרת $\wedge$ השני ($\wedge$ elimination)

$$\frac{\phi \wedge \psi}{\psi} \wedge e_2$$

- $\phi \wedge \psi$ - נתון
- ψ - מסקנה
- $\wedge e_2$ – שם הכלל (elimination = e)

שימוש הכלל בהוכחה:

$$p \wedge q, r \vdash q \wedge r$$

1	$p \wedge q$	נתון
2	r	נתון
3	q	$\wedge e_2$ 1
4	$q \wedge r$	$\wedge i$ 2, 3

שם הכלל ועל איזה שורה או שורות הוא מופעל

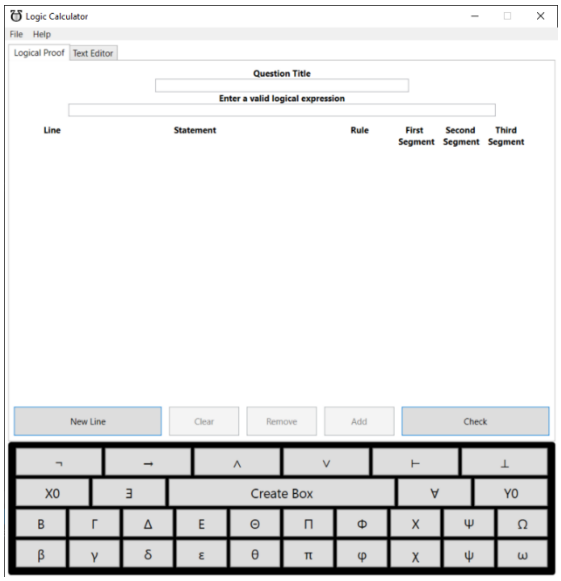
Design

התוכנה מחולקת לשלוש מחלקות עיקריות:

Static Graphic user interface

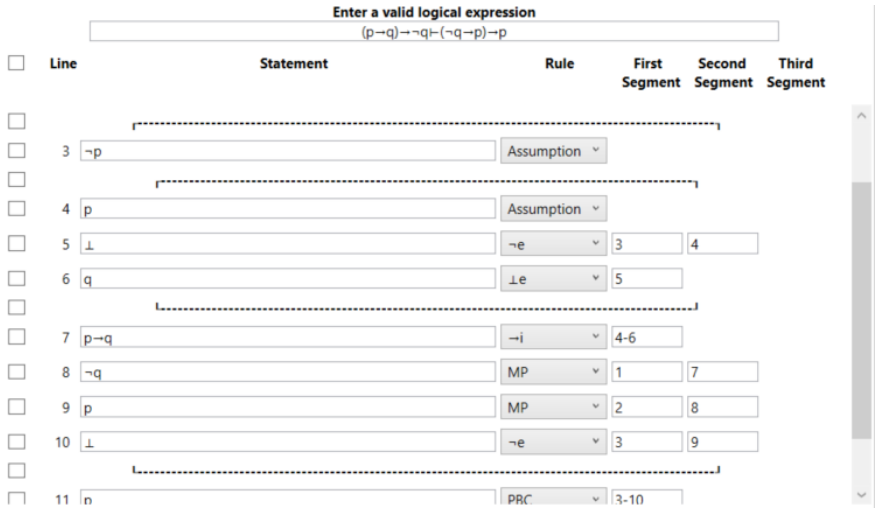
חלק זה נבנה בתחילת הרצת התוכנה ואינו משתנה לכל אורך ריצת התוכנה.

זהו השלד של הGUI המוצג למשתמש. למשל: מקלדת, תפריט, טאבים.



Dynamic Graphic user interface

חלק זה תלוי בצרכי המשתמש, מאפשר שליטה למשתמש בכל מה שקשור בשורות ההוכחה, מאפשר הוספת/מחיקה/ניקוי שורות, כמו כן מתאים את עצמו לפי בחירת החוק הנבחר על ידי המשתמש בשורה הספציפית (משפיעה על כמות הסגמנטים המאותחלים בשורה). חלק זה גם אחראי על יצירת תיבות ההוכחה.

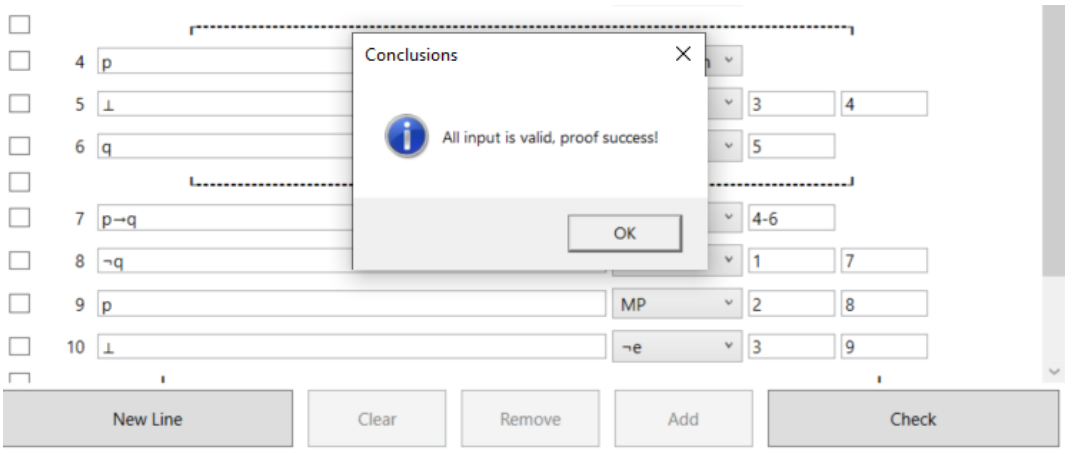


Evaluation

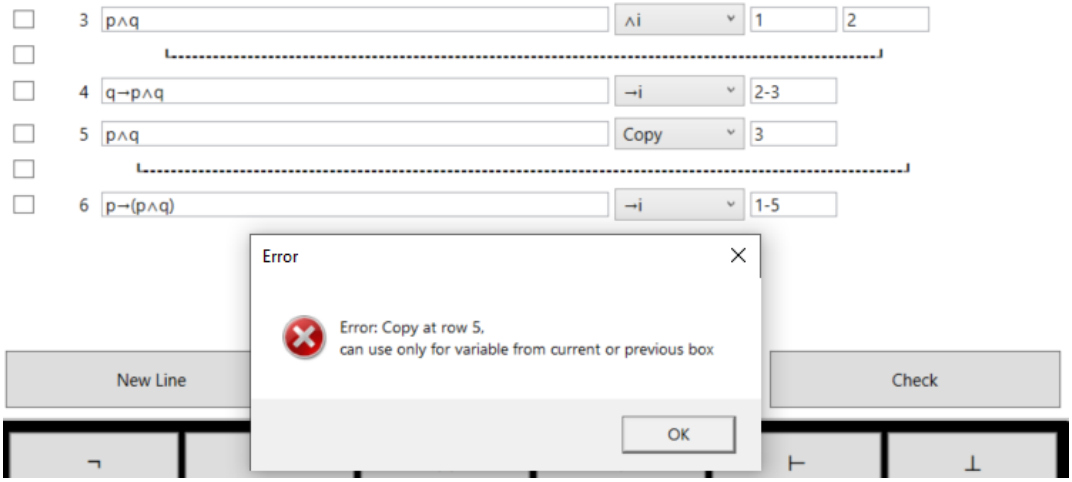
חלק זה של התוכנה הוא המהות שלה. בקוד עצמו הוא מופיע במחלקה נפרדת, שמכילה בתוכה את מימושי חוקי הלוגיקה על פי ספר הקורס :

Huth M., Ryan M., Logic in Computer Science, 2nd ed. Cambridge 200

ספר זה הוא מקור לימוד מקובל במקומות שונים. החלק הזה אחראי על בדיקת הנכונות של ההוכחה שנכתבה על ידי המשתמש עד השלב הנוכחי. הוא נכנס לשימוש רק מרגע לחיצה על כפתור "Check", על מנת שהשליטה תהיה בידי המשתמש. ראשית אנחנו בודקים את הסינטקס של הקלט ולאחר מכן את הנכונות הלוגית של ההוכחה, למשל, מספרי שורות, מיקום התיבות וכולי.



במקרה של בעיה בהוכחה, תקפוץ הודעה למשתמש שמכילה פרטים על מיקום הבעיה ומה הטעות שקרתה.



בניית הפרויקט - ציר זמן

אפיון ראשוני

בתחילה דיברנו עם פרופ׳ דני לגבי מה הבעיות והצרכים שעלינו לענות עליהם, ואיזה פונקציונליות הוא מעוניין שתהיה לתוכנה. לאחר מכן, ישבנו ויצרנו סקיצה ראשונית של מבנה התוכנה, ניסינו להבין כיצד להמיר את המבנה הברור של הוכחה לוגית לכדי תוכנה שמבוססת WPF.

רצינו לשים את נוחות השימוש במקום הראשון, שכן זאת אחת המטרות העיקריות של הפרויקט, ולבנות מודל אשר יהיה ברור למשתמש, זאת על ידי שמירה על עיצוב הדומה להוכחה לוגית סטנדרטית.

העיצוב נבנה בראשינו כאשר הסתכלנו על המחשב הנייד שלנו וממנו נלקח הרעיון של יצירת המקלדת מתחת למסך הראשי, כאשר מרכז התוכנה יהווה את החלק העיקרי בו המשתמש יבצע את פעולותיו ואילו בחלק העליון נוסיף אפשרויות עזר.

בנייה סטאטית

בשלב זה בנינו את דף הXAML, שהוא בעצם השלד של התוכנה, מטרת העל של השלב הזה הייתה להגיע לאב טיפוס ראשוני בכל מה שקשור בתצורת התוכנה, הרצון היה ליצר כמה שיותר מהר טופס סטאטי אשר יהווה מסגרת כלשהי עליה נוכל להתחיל לבנות ולהרחיב.

מכיוון שרוב העבודה בפרויקט הייתה בזמן הקורונה, וחלק נכבד היה תוך כדי עבודה ולימודים של שנינו, החלטנו לפצל את העבודה על מנת לפתח במקביל את התצורה הדינאמית של התוכנה ותוך כדי להתחיל במימוש חוקי תחשיב היחסים. החלטנו לממש את הקליטה של נימוקי ההוכחה בעזרת עד שלושה ׳טקסט בוקסים׳ שאותם כינינו סגמנטים, כדי להקל עלינו בקליטה שלהם.

מימוש חוקי תחשיב הפסוקים

לפני תחילת המימוש של בדיקות החוקים הראשונים, החלטנו להפריד לגמרי בין מה שקשור ל GUI והבניה שלו לבין בדיקות החוקים והקלט של המשתמש. זה אפשר לנו לסדר את הקוד בצורה יותר נוחה לנו וגם עבודה יותר פשוטה עם הגיט באמצעות שני קבצים שונים. כך יכולנו לעבוד במקביל ולדחוף שינויים רבים בלי להפריע אחד לשני. כך נוצרה מחלקת Evaluation אשר הכילה בתוכה את המימושים הראשונים של בדיקות החוקים. לאחר מכן חזרנו על החומר בקורס וניסינו להבין מהי הדרך האופטימלית לממש את הבדיקות כך שהן יאפשרו למשתמש חופש פעולה מרבי תוך כדי שמירה על נכונות ההוכחה.

כדי להעביר את תוצאת הבדיקה הוספנו במחלקה זו משתנה בשם Is_Valid, כאשר אם ערכו בסוף בדיקה הוא שלילי על חוק כלשהו, אז התוכנית תתריע על השגיאה. השתדלנו לתת את כמות המידע המרבית למשתמש על כל שגיאה פוטנציאלית.

בשלב זה נתקלנו בהתלבטות משמעותית, הוכחה לוגית נבנית בצורה אינדוקטיבית, כלומר שבכל שורה שאותה המשתמש כותב ההנחה היא שהשורות לפניה **נכונות**, לכן התלבטנו האם עלינו

לבדוק בכל פעם רק את השורה האחרונה שנוספה על ידי המשתמש או לעבור מהתחלה על כל שורות ההוכחה.
לבסוף החלטנו לבצע תמיד בדיקה על כל שורות ההוכחה מאחר ואנחנו נותנים למשתמש לבצע שינויים בכל שורה בה הוא רוצה ולכן אין הבטחה כי שורות ישנות שהוא הכניס לא עברו שינוי על ידו ועל כן יש לבדוק גם אותן. המשתמש מחליט את הכמות והתזמון של הבדיקות בעצמו באמצעות כפתור ייעודי.

בנייה דינאמית
המרת האבטיפוס ובנייתו בצורה דינאמית, זהו היה שלב שדרש מאיתנו השקעה של הרבה זמן במחשב והוספת קוד.
יש לציין שבקורס ב #C לא נוגעים באופן בניית GUI דינאמי בכלל והיינו צריכים להבין כיצד ניתן לייצר אובייקטים חדשים של WPF ולאחר מכן לבצע עליהם מניפולציות בזמן אמת, באמצעות הקוד שכתבנו.
רצינו לייצר כמה שיותר אפשרויות למשתמש לשינוי, עריכה והוספה של שורות ההוכחה וכמו כן להגביל אותו בשימוש בכמות סגמנטים רלוונטית לאחר בחירת החוק בו הוא רוצה להשתמש.
בשלב זה גם הכנסנו את כל השליטה בתיבות ההוכחה (Boxes) ויצירתן.

לקראת סוף שלב זה ביצענו Demo ראשון בפני דני, הצגנו לו את היכולות הנוכחיות, התייעצנו לגבי קשיים שצצו במהלך הדרך, והתדיינו לגבי המשך הדרך.

שמירה וטעינה
הוספנו ממשק בין התוכנה לקבצי Word בעזרת שימוש בחבילת "**Xceed docs**" אשר נתנה לנו API עם יכולות יצירה וטעינת מסמכים.
היה לנו ברור ששלב זה קריטי וחשוב מאחר ומטרתה הסופית של התוכנה היא לעזור בהגשת שיעורי הבית ולכן היה לנו ברור שצריך לאפשרות לייצר פלט מסודר ומאורגן אותו הסטודנטים יוכלו להגיש.
בנוסף אפשרות הטעינה מאפשר למשתמש להפסיק את שיעורי הבית באמצע ולחזור אליהם בשלב מאוחר יותר וכמו כן מאפשר לבודק התרגילים לטעון את תרגילי הסטודנטים ולבצע בדיקה מהירה על נכונות התרגיל.
כמובן ששלב זה גם עזר לנו בהמשך בכל שלב הDebug ונתן לנו אפשרות לשמור ולטעון הוכחות אשר נמצאו בהן באגים, ולחסוך כמות עצומה של זמן על כתיבת הוכחות שוב ושוב במקרה שהתוכנית קורסת.

מימוש חוקי תחשיב היחסים

בשלב זה הרחבנו את מחלקת Evaluation עם מימושים לבדיקות של חוקים החלים על פרדיקטים. בנוסף לקושי הבסיסי של חזרה על החוקים האלו והשימוש בהם, שהוא משמעותית פחות אינטואיטיבי מהחוקים הבסיסים, נתקלנו בבעיות חדשות מבחינת בדיקות קלט. למשל, בתוך יחס או פונקציה יכולים להתקבל משתנים רבים המופרדים בפסיק, אז הוספנו בדיקה שמוודא שאם המשתמש הכניס יחס/פונקציה עם מספר מסוים של משתנים הוא לא יוכל להכניס את אותה הפונקציה עם מספר אחר של משתנים. כמו כן, יחס יכול לקבל פונקציות בתור ארגומנטים מה שמסבך את הבדיקות של הקלט.

שימוש בהוכחה ידועה

בעקבות הצעתו של דני הוספנו את היכולת להשתמש בהוכחה שכבר הוכחה כנכונה בהוכחה הנוכחית, הדבר מאפשר למשתמש לוותר על שלבים בהוכחה במידה ויש משהו שהוכח כבר בעבר ועל ידי כך לקצר את ההוכחה.

חוק זה הינו **גנרי** ומקל על שימוש המשתמש.

☐	Line	Statement	Rule	First Segment	Second Segment	Third Segment
☐	1	δ∧β,γ⊢γ→β	Proven i			
☐	2	r	Data			
☐	3	(p∨r)∧q	Data			
☐	4	r→q	Proven e	1	3,2	

Debug

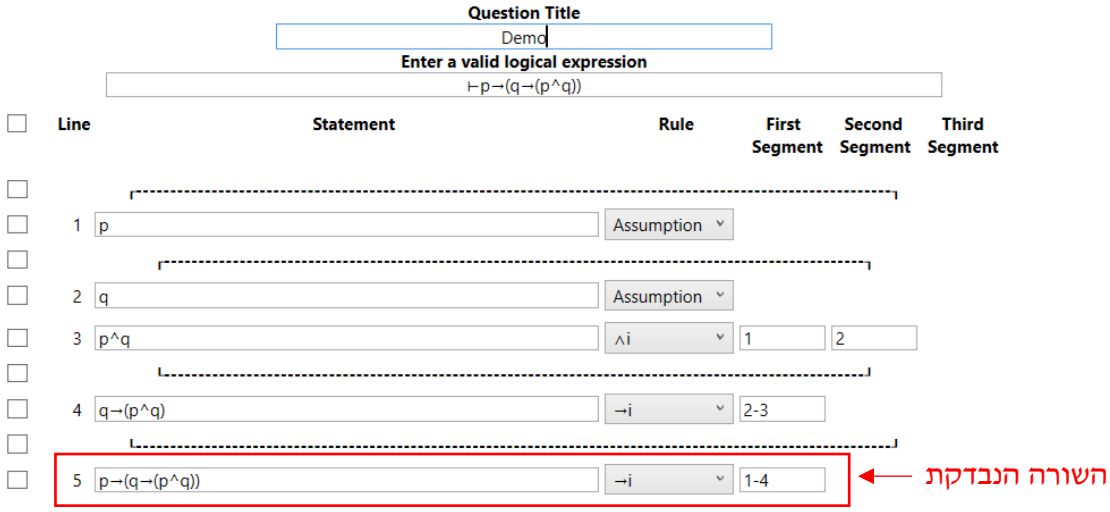
בסיום שלב הבניה של התוכנה, כאשר החלטנו שהגענו ליעד שהצבנו לעצמנו וכל ה Features הוכנסו, התחלנו לבצע בדיקות מקיפות ולחפש אילו חוקים אינם עובדים כראוי ובאילו מקרים. מאחר והמטרה הסופית שלנו היא שהתוכנה תהיה בשימוש של סטודנטים היה עלינו לנסות למצוא ולסגור כמה שיותר באגים על מנת להוציא תוכנה כמה שיותר אמינה.

השקענו שעות רבות בשימוש בתוכנה, השתמשנו בתרגילים ממבחנים, שיעורי בית וספר הקורס על מנת לנסות לראות כמה שיותר מצבי קיצון.

אלגוריתמיקה

אופן פעולת הבדיקה של התוכנה ממומש בצורה אינדוקטיבית, אנו בודקים את ההוכחה מהשורה הראשונה ועד האחרונה בצורה איטרטיבית , בצורה זו אנו יכולים להיות בטוחים שאם אנו מסתמכים בשורה הנוכחית על משהו שנגזר בשורות הקודמות הוא נבדק והוא נכון.

נציג בקטע זה את האלגוריתם לבדיקת נכונות שורה, נשתמש בדוגמא על שורה ספציפית אך אופן הפעולה זהה לכל שורה בהוכחה.



בדיקת נכונות שורת הוכחה מתחלקת לשלושה חלקים:

קלט

בתחילה אנחנו בודקים את הביטוי הראשי שמוכנס מעל הטבלה.

לאחר שמוודאים שהוא תקין אנו מתחילים 'לקורא' את קלט המשתמש לפי סדר השורות.
מהקלט בכל שורה יוצרים אובייקט Statement שמכיל את כל שדות השורה הנוכחית:

expression, rule, first_segment, second_segment, third_segment

```
expression = Utility.ReplaceAll(((TextBox)row.Children[STATEMENT_INDEX]).Text);
rule = Utility.ReplaceAll(((ComboBox)row.Children[COMBOBOX_INDEX]).Text);
first_segment = ((TextBox)row.Children[SEGMENT1_INDEX]).IsEnabled ? Utility.ReplaceAll(((TextBox)row.Children[SEGMENT1_INDEX]).Text) : null;
second_segment = ((TextBox)row.Children[SEGMENT2_INDEX]).IsEnabled ? Utility.ReplaceAll(((TextBox)row.Children[SEGMENT2_INDEX]).Text) : null;
third_segment = ((TextBox)row.Children[SEGMENT3_INDEX]).IsEnabled ? Utility.ReplaceAll(((TextBox)row.Children[SEGMENT3_INDEX]).Text) : null;
statement_list.Add(new Statement(expression, rule, first_segment, second_segment, third_segment));
```

main_expression	"⊢p→(q→(p∧q))"	string
expression	"p→(q→(p∧q))"	string
rule	"→i"	string
first_segment	"1-4"	string
second_segment	null	string
third_segment	null	string
index	8	int

לאחר מכן מגיע שלב בדיקת הסינטקס של ה Expression בשורה (שימוש בסוגריים, שימוש בתווים מותרים, שימוש נכון במשתנים ובסימנים הלוגיים וכיוצא בזה..)

```
private bool IsValidStatement(string expression, string rule, string first_segment,
                              string second_segment, string third_segment)
{
    int row = statement_list.Count-1;
    if (!IsValidExpression(expression, row+1))
    {
        return false;
    }
    if (string.IsNullOrEmpty(rule))
    {
        Expression_Error(row+1, "Rule is empty");
        return false;
    }
    int ret;
    if (first_segment != null)
    {
        if (first_segment == string.Empty)  ≤ 1ms elapsed
        {
            Expression_Error(row, "First segment is empty");
            return false;
        }
        else if ((ret = IsValidSegment(first_segment, row+1)) != 0)
        {
            if (ret == -ERRARGUMENT)
                Expression_Error(row, "First segment is not a positive integer number");
            else if (ret == -ERRMISSINTEGER)
                Expression_Error(row, "Missing positive integer in first segment");
            else if (ret == -ERRINDEX)
                Expression_Error(row, "First segment index must be smaller then row number");
            else if (ret == -ERRCOMMA)
                Expression_Error(row, "Comma is only allowed in Proven e segments");
            return false;
        }
    }
```

בשלב הבא נבדקים ה Segments (הרלוונטיים לחוק), מתבצעת בדיקה שהשורות שאליהן המשתמש הפנה אכן קיימות, ההפניה היא רק לשורה קודמות לשורה הנוכחית ואין שימוש בסימנים אסורים.

```
public int IsValidSegment(string seg, int currentRow)
{
    string[] splitted = seg.Split(new Char[] { ',', '-' });
    if (seg.Contains('-') && (string.IsNullOrEmpty(splitted[1]) || string.IsNullOrEmpty(splitted[0])))
        return -ERRMISSINTEGER;
    if (seg.Contains(','))
    {
        if (statement_list[currentRow - 1].Rule != "Provene")
            return -ERRCOMMA;
        foreach (string s in splitted)
        {
            if (string.IsNullOrEmpty(s))
                return -ERRMISSINTEGER;
        }
    }
    foreach (var s in splitted)
    {
        if (!Int32.TryParse(s, out _))
            return -ERRARGUMENT;
        if (int.Parse(s) >= currentRow)
            return -ERRINDEX;
    }
    return SUCCESS;
}
```

הודעת שגיאה מפורטת תוצג למשתמש במקרה שאחת הבדיקות יוצאת שלילית.

שימוש בחוק

כעת אנו מוסיפים את השורה הנוכחית לתוך רשימה של Statements. את הרשימה הזאת אנחנו מעבירים עם החוק שנבחר בשורה הנוכחית למחלקת Evaluation, תפקידה של המחלקה היא בדיקת הנכונות הלוגית על פי החוק שנבחר על ידי המשתמש, במקרה של הדוגמא הנתונה "→i".

```
internal class Evaluation
{
    99+ references
    public bool Is_Valid { get; set; }
    private readonly List<Statement> statement_list;
    private readonly int current_line;
    1 reference
    public Evaluation(List<Statement> statement_list, string rule)
    {
        Is_Valid = false;
        this.statement_list = statement_list;
        current_line = statement_list.Count - 1;
        Handle_Rule(rule);
    }
    #region RULES
    1 reference
    private void Handle_Rule(string rule)
    {  ≤ 1ms elapsed

        switch (rule)
        {
            case "→i":
                Arrow_Introduction();
                break;
            case "X0/Y0i":
                Var_Introduction();
                break;
```

rule "→i" string

הפונציה Handle_Rule אחראית להעביר את טיפול השורה לפונקציה רלוונטית על פי סוג החוק.

בדיקת Arrow_Introduction עושה Parsing על שורת הסגמנטים של החוק ולוקחת את הStatements המופיעים בשורות הרלוונטיות (1), בודקת הופעת "→" בStatement הנוכחי (2) ומוודא כי אכן הStatement הוא הרכבה של Statements מגבולות התיבה (3).

במידה ואחת הבדיקות נכשלת הודעת שגיאה מתאימה תופיע למשתמש.

```
private void Arrow_Introduction()
{
    int last_line = Utility.Get_Last_Line_From_Segment(statement_list[current_line].First_segment, current_line);
    Is_Valid = last_line != -1;
    if (!Is_Valid)
    {
        return;
    }
    int first_line = Utility.Get_First_Line_From_Segment(statement_list[current_line].First_segment, current_line);
    Is_Valid = first_line != -1;
    if (!Is_Valid)
    {
        return;
    }

    string current_expression = statement_list[current_line].Expression,
        start_expression = statement_list[first_line].Expression,
        end_expression= statement_list[last_line].Expression;

    Is_Valid = current_expression.Contains("→");
    if (!Is_Valid)
    {
        Utility.DisplayErrorMsg("Missing → in 'arrow introduction", current_line);
        return;
    }
    Is_Valid = statement_list[first_line].Rule == "Assumption";
    if (!Is_Valid)
    {
        Utility.DisplayErrorMsg("The first segment provided in arrow introduction must start with assumption", current_line);
        return;
    }

    Is_Valid = current_expression == start_expression + "→" + end_expression
            || current_expression == "(" + start_expression + ")→" + end_expression
            || current_expression == start_expression + "→(" + end_expression + ")"
            || current_expression == "(" + start_expression + ")→(" + end_expression + ")";

    if (!Is_Valid)
    {
        Utility.DisplayErrorMsg(current_expression+ " should be equal to "+ start_expression + "→" + end_expression, current_line);
    }
}
```

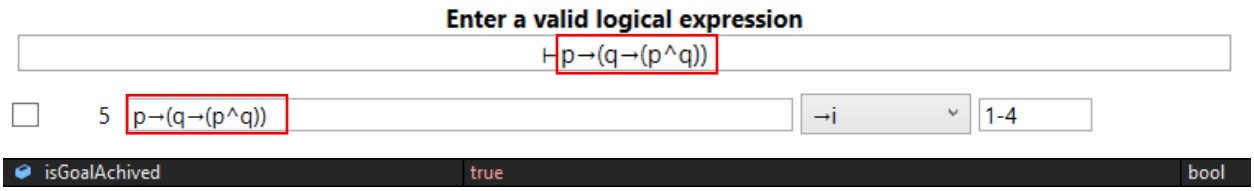
תיבות הוכחה

מנגנון בדיקת תיבות ההוכחה מופרדת מהבדיקות הקודמות אך אנו מתייחסים לחלק זה כחלק בלתי נפרד מאלגוריתם הבדיקה.

מאחר והתיבות (Boxes) מיוצרות על ידי הGUI הוא **אינו** נחשב כחלק מקלט המשתמש.

בעקבות החלטה נוספת בביצוע התוכנית ואופן יצירת התיבות בתוכנה החלטנו שבדיקות נכונות השימוש בתיבות ההוכחה תתבצע רק כאשר המשתמש הצליח להוכיח את מה שהוא רצה.

בדיקות תיבות ההוכחה מתחילות אם ורק אם כל השורות בהוכחה נכונות מבחינה סינטקטית ולוגית, וגם ההוכחה הגיע לסיומה.



במצב זה אנו נכנסים לאיטרציה נוספת על כל חלקי ההוכחה בה נבדקת אך ורק נכונות השימוש בתיבות ההוכחה, כמובן רק בחוקים אשר יש לתיבות משמעות.

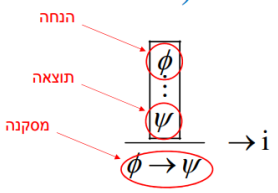
```
bool isGoalAchived = CheckGoalAchieved(spGridTable.Children[spGridTable.Children.Count - 1] as Grid);
if (isGoalAchived)
{
    foreach (Grid row in spGridTable.Children)
    {
        index = spGridTable.Children.IndexOf(row);
        if (row.Children[LABEL_INDEX] is Label)
        {
            expression = Utility.ReplaceAll(((TextBox)row.Children[STATEMENT_INDEX]).Text);
            rule = Utility.ReplaceAll(((ComboBox)row.Children[COMBOBOX_INDEX]).Text);
            first_segment = ((TextBox)row.Children[SEGMENT1_INDEX]).IsEnabled ? Utility.ReplaceAll(((TextBox)row.Children[SEGMENT1_INDEX]).Text) : null;
            second_segment = ((TextBox)row.Children[SEGMENT2_INDEX]).IsEnabled ? Utility.ReplaceAll(((TextBox)row.Children[SEGMENT2_INDEX]).Text) : null;
            third_segment = ((TextBox)row.Children[SEGMENT3_INDEX]).IsEnabled ? Utility.ReplaceAll(((TextBox)row.Children[SEGMENT3_INDEX]).Text) : null;

            if (!IsValidBox(row, rule, first_segment, second_segment, third_segment))
                return;
        }
    }
```

בתוך הפונקציה IsValidBox אנו בודקים אם מתבצע השימוש בתיבות ההוכחה כמו שהחוק דורש

```
case "¬i":
case "→i":
case var a when new Regex(@"∀.*i").IsMatch(a):
case var e when new Regex(@"∃.*e").IsMatch(e):
    string segment = "first";
    List<int> box;
    if (new Regex(@"∃.*e").IsMatch(rule)&&second_segment.Contains("-"))
    {
        box = Utility.Get_Lines_From_Segment(second_segment);
        segment = "second";
    }
    else
        box = Utility.Get_Lines_From_Segment(first_segment);
    if (!HasWrapBox(box[0], box[box.Count - 1]))
    {
        string messageRule = "rule";
        if (rule.Equals("¬i"))
            messageRule = "Not introduction";
        else if (rule.Equals("→i"))
            messageRule = "Arrow introducstion";
        else if (new Regex(@"∀.*i").IsMatch(rule))
            messageRule = "All introdution";
        else if (new Regex(@"∃.*e").IsMatch(rule))
            messageRule = "Exist elimination";
        Utility.DisplayErrorMsg($"Error: {messageRule} at line {((Label)row.Children[LABEL_INDEX]).Content}," +
            $"\nlines mentioned in {segment} segment must be wrapped with box");
        return false;
    }
    return true;
```

במקרה של הדוגמא שלנו:

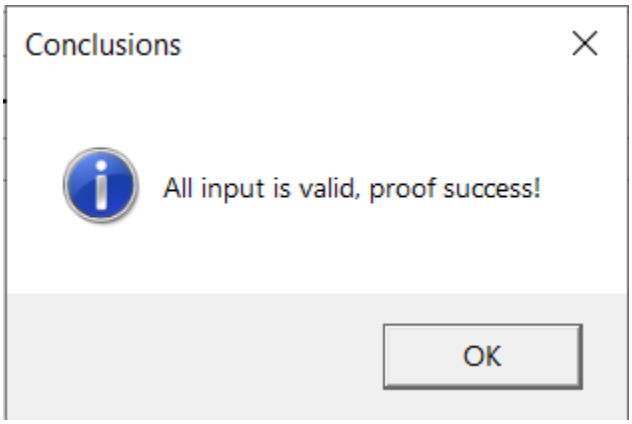
כלל הוספת גרירה (→ introduction)



אנו צריכים לוודא כי ישנה תיבה אשר עוטפת את השורות אליהן הפנה המשתמש,
בדיקה זו מבוצעת ע"י הפונקציה HasWrapBox.

```
private bool HasWrapBox(int open, int close)
{
    int realOpenIndex = GetspGridIndex(open),
        realCloseIndex = GetspGridIndex(close);
    if (realOpenIndex <= 0 || realCloseIndex < 0 || realCloseIndex == spGridTable.Children.Count - 1)
        return false;
    if (!IsCorrectBox(realOpenIndex - 1, BoxState.Open) || !IsCorrectBox(realCloseIndex + 1, BoxState.Close))
        return false;
    return true;
}
```

בסופו של התהליך, אם כל הבדיקות עברו המשתמש יקבל הודעה אשר תודיע לו שהכל תקין והוא סיים את ההוכחה.



תוצר סופי

- כלי בדיקת נכונות הוכחה שנכתבה על ידי המשתמש מתריע במידה וישנו כשלון על מיקומו ואופיו.
- תוכנה בעלת ממשק עבודה ברור ואינטואיטיבי למשתמש, הממשק דומה מאוד לכל תוכנת Office ומקל מאוד על משתמשים אשר נחשפים לתוכנה בפעם הראשונה, מבנה הכנסת הקלט בכלי הינו זהה לתצורת כתיבת ההוכחה הנלמדת בקורס.
- ספרון עזרה משתמש (User manual) מפורט, בו הסברים מפורטים עם תמונות על שימוש נכון בתוכנה, יכולותיה והסברים על השימוש בכל אחד מהחוקים.
- קובץ התקנה פשוט ומהיר אשר מאפשר לסטודנט התקנה קלה ונוחה ושימוש מידי על כל מחשב בעל מערכת הפעלה Windows.
- אפשרות שמירה/טעינה מסמכי Word הופכת את התוכנה לרלוונטית וקלה לתפעול בכל הנוגע להגשת שיעורי הבית של הסטודנט, ובדיקת התרגילים.
- מחלקת Evaluation אשר מכילה בתוכה את מימוש כל החוקים ניתנת להרחבה בקלות, ומאפשרת את הרחבת הפרויקט במידה ויש צורך בכך.
- תצורת הודעת שגיאה למשתמש מאפשרת לו להבין בצורה מפורטת את הטעות אותה ביצע ביצירת ההוכחה כמו גם את מיקומה המדויק, אך בו בזמן אין התוכנה חושפת את הדרך הנכונה לפתרון.
- הפרויקט נוהל מתחילתנו ועד סופו תחת Github, דבר אשר מאפשר גישה מכל מקום ומכל מחשב בעל חיבור לאינטרנט וכמו כן אפשרות עתידית להרחבת הפרויקט ושימוש בSource code שיצרנו.

סיכום

קיימים שלל סוגי "מחשבוני לוגיקה" שנותנים פתרון מידי להוכחה ו/או מייצרים טבלת אמת, התוכנה אותה יצרנו הותאמה במיוחד לצרכי הסטודנטים ונותנת פתרון שאינו מן הנמצא לבעיה קיימת.

הפרויקט היה חשוב לנו מאוד, התנסנו בזכותו בתהליך שלם של יצירת תוכנה, החל מרעיון ראשוני, מימוש, UI, UX, שימוש נרחב בשפת #C, Debugging אינטנסיבי, ניהול קוד מלא בעזרת Git ולבסוף Publishing.

נעזרנו לכל אורך התהליך בפרופסור דני קוטלר, לקחנו את המלצותיו לתשומת לבינו וניסינו לייצר תוכנה נוחה ורלוונטית.

ניסינו כמה שיותר לשים את עצמנו במקומו של המשתמש ולייצר עבורו חוויה טובה עד כמה שניתן, מתוך הניסיון שלנו בתור סטודנטים באופן כללי, ובתור כאלו שעברו את הקורס לוגיקה באופן ספציפי.

אנחנו מרגישים שנתנו את המירב בפרויקט והתייחסנו אליו בכבוד הראוי, ואנחנו מרגישים תחושת סיפוק רב מהתוצאה.

אנו מקווים שהתוכנה אכן תממש את מטרתה העיקרית ותהוה כלי בשימוש קבוע על ידי תלמידי קורס לוגיקה במכללה בשנים הבאות, ותסייע להם בלימוד הקורס.

הפרויקט היווה עבורנו סוג של סגירת מעגל, בתחילת התואר למדנו את הקורס מבוא למדעי המחשב בשפת C שהעביר דני כשהיה ראש החוג, וכעת אנחנו מסיימים את התואר עם פרויקט בשפת ב#C שמנחה דני שעכשיו נמצא בתפקיד דיקן המדעים.

אנו מודים לדני על התמיכה והליווי לכל אורך הפרויקט והתואר.